An exact algorithm for sparse matrix bipartitioning*

Daniël M. Pelt† and Rob H. Bisseling‡

Abstract

The sparse matrix partitioning problem arises when minimizing communication in parallel sparse matrix-vector multiplications. Since the problem is NP-hard, heuristics are usually employed to find solutions. Here, we present a purely combinatorial branch-and-bound method for computing optimal bipartitionings of sparse matrices, in the sense that they have the lowest communication volume out of all possible bipartitionings obeying a certain load balance constraint. The method is based on a way of partitioning similar to the recently proposed medium-grain heuristic, which reduces the number of solutions to be considered in the branch-and-bound method.

We applied the proposed optimal bipartitioner to find the optimal communication volume of all matrices of the University of Florida sparse matrix collection with 1000 nonzeros or less. For 85% of the matrices, an optimal bipartitioning was found within a single day of computation and for 58% even within a second. We also present optimal results for selected larger matrices, up to 129,042 nonzeros. The optimal bipartitionings and corresponding communication volumes are made publicly available in a benchmark collection.

*This paper has been published in *Journal of Parallel and Distributed Computing*, vol. 85 (2015), pp. 79-90, doi:10.1016/j.jpdc.2015.06.005

†Scientific Computing Group, Centrum Wiskunde & Informatica, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands, D.M.Pelt@cwi.nl

‡Mathematical Institute, Utrecht University, P.O. Box 80010, 3508 TA Utrecht, The Netherlands, R.H.Bisseling@uu.nl

1 Introduction

Parallel iterative linear system solvers can be tremendously accelerated by using a good partitioning of the sparse matrices involved, which means that the parts have nearly equal size and that there are few dependencies between them. Other parallel computations that are based on sparse matrix–vector multiplication (SpMV), such as eigensystem solvers, can benefit as well.

The *sparse matrix partitioning problem* can be defined as finding a partitioning of a sparse $m \times n$ matrix A with N nonzeros in p disjoint parts,

$$A = \bigcup_{i=0}^{p-1} A_i, \tag{1}$$

such that the number of nonzeros of part A_i satisfies

$$|A_i| \leq (1+\varepsilon) \left\lceil \frac{N}{p} \right\rceil, \quad \text{for } 0 \leq i < p, \tag{2}$$

where $\varepsilon \geq 0$ is a given load-imbalance parameter, and such that the communication volume in the corresponding parallel SpMV is minimized. The load balance constraint (2) is formulated such that the extreme case $\varepsilon = 0$ still has a feasible solution. The *communication volume* of a matrix column j is defined as $\lambda_j - 1$, where λ_j is the number of matrix parts with a nonzero in column j. This volume occurs because in a parallel SpMV,

$$\vec{u} = A\vec{v}, \tag{3}$$

we have to send input vector component v_j to all parts that have a nonzero in column j, except for one part, provided we assign vector component v_j to one of the λ_j parts. This communication is shown as vertical arrows in Figure 1. Similarly, we can define the communication volume of a matrix row. The total communication volume $Vol = Vol(A_0, \ldots, A_{p-1})$ is then the sum of the communication volumes of all rows and columns, and our optimization objective is to minimize Vol.

Finding an optimal sparse matrix partitioning is NP-hard, even for $p = 2$, because the underlying hypergraph partitioning problem is NP-hard [31]. Therefore, most solution methods so far have been heuristic, trying to find a good but not necessarily optimal partitioning in reasonable time. An example of a fast heuristic is the medium-grain method [34] which we recently developed; this method will be briefly explained in Section 3.

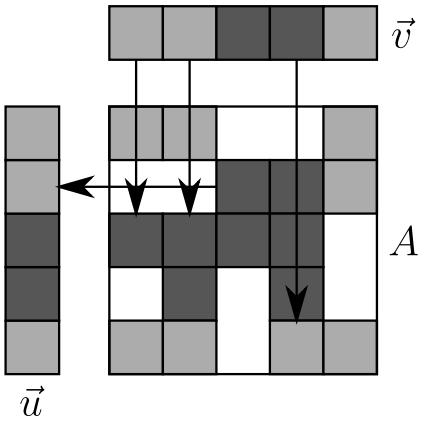


Figure 1: Parallel multiplication of a 5×5 sparse matrix A and a dense input vector $\vec{v}$ giving a dense output vector $\vec{u} = A\vec{v}$. The 16 nonzero elements of A have been partitioned and assigned to $p = 2$ processors, depicted in light and dark gray. The vector components have also been assigned to these two processors. The parallel computation starts by communicating three vector components v_j, as depicted by vertical arrows, then it computes and adds all products $a_{ij}v_j$ locally, and finally it sends one contribution, for the second row of A, as depicted by a horizontal arrow, to enable computation of the output components $u_i = \sum_i a_{ij}v_j$. Note that the other rows do not require communication. The total communication volume is $\mathit{Vol} = 4$. The load balance of the nonzeros is perfect ($\varepsilon = 0$).

The purpose of the present article is to find an optimal solution, accepting much longer computation times, and if needed limiting the size of the problems we can solve. Our motivation is that having a suite of problems with optimal partitionings will be useful, because we can then compare heuristic solutions with an optimal benchmark solution, and see how good the heuristic methods really are. As heuristics are improving, perhaps even to the point of saturation, it may be beneficial to know how far we are from an optimal solution, and perhaps decide to optimize further for other, secondary objectives instead, such as the total number of messages sent.

In this article, we will concentrate on bipartitioning, i.e. $p = 2$, because this is the easiest problem and we can expect to build a larger suite of solved problems than for $p > 2$, and also because many partitioners are based on recursive bipartitioning. The resulting suite will be made available through the website of the Mondriaan package[1]. Furthermore, we present in this article and on the website a set of pictures of optimal solutions for small matrices. In our experience, visualization of optimal partitionings is not only pleasing to the eye, but also helpful in inspiring new ideas for improving current heuristic solution methods. As a matter of fact, this is how we were led to design the medium-grain method [34].

2 Related work

Çatalyürek and Aykanat [10] were the first to formulate the minimization of the communication volume of a parallel SpMV as a hypergraph partitioning problem, thus solving the problem in the correct metric. Previously, graph partitioning was commonly employed, which only gives an approximation of the correct volume. In the *row-net model* of Çatalyürek and Aykanat, the n columns of the sparse matrix are modeled by the vertices of a hypergraph, and the m rows are modeled by nets (hyperedges, i.e. subsets of the vertices), such that vertex i is contained in net j if and only if $a_{ij} \neq 0$. The balance criterion of Eq. (2) is translated into a criterion on the weights of the vertices, where the weight of vertex j is defined as the number of nonzeros in matrix column j. The communication volume is modeled as the sum of the costs $\lambda_i - 1$ for all nets (rows) i. In the *column-net model*, the roles of rows and columns are reversed. Both models yield a one-dimensional (1D) matrix partitioning.

[1] `http://www.staff.science.uu.nl/~bisse101/Mondriaan/`

A different model by the same authors is the *fine-grain model* [11], which is two-dimensional (2D) in nature. It models the N nonzeros as vertices in a hypergraph and it has both m row nets and n column nets, defined similarly as in the 1D case. This model also minimizes the correct volume, and since it is more general it can in principle achieve better solutions; this is at the cost of longer computation times and more memory usage, as the hypergraph has many more vertices.

A different 2D method for $p > 2$ can be obtained by repeatedly bipartitioning a submatrix of A, trying both 1D hypergraph models with $p = 2$, and using the best of the two, which is done in the earlier versions of the Mondriaan package [38] (until version 3), and which we call the *localbest* method. In the latest version (version 4) of Mondriaan, the default has been changed to the recent *medium-grain* method.

Communication volume may not be the only relevant metric for the actual communication time of a parallel SpMV. Boman, Devine, and Rajamanickam [6] present a 2D method based on combining 1D graph/hypergraph partitioning with a 2D block distribution that also limits the total number of messages, besides trying to minimize the communication volume. A different approach to minimize other metrics as well is taken by the authors of the UMPa package [13], where the total and maximum volume per processor, and the total and maximum number of messages per processor can be chosen as primary or secondary objectives, and the secondary objective is used to break ties. This necessitates the use of a *directed hypergraph*, where every net has a source vertex.

Several software packages for hypergraph partitioning are currently available: sequential packages hMetis [25], PaToH [10], Mondriaan [38], and the parallel packages Par*k*way [37] and Zoltan [17]. Zoltan also contains a parallel toolkit Isorropia [5] that provides a sparse matrix partitioning interface (currently only for 1D partitioning). All these partitioners are heuristic, and all are based on a multilevel approach, first coarsening the hypergraph to obtain a smaller hypergraph that still resembles the original one, then obtaining an initial partitioning, and finally projecting back the solutions during the uncoarsening, while further refining them.

Graph partitioners have been studied for at least four decades, with the seminal paper by Kernighan and Lin [28] providing one of the first heuristic algorithms. The graph partitioning problem is usually defined as partitioning the vertices of a graph in such a way that the number of vertices is balanced with an allowed imbalance fraction of ε, similar to Eq. (2), and the objective

is to minimize the total *edge cut*, the number of edges of the graph with vertices in different parts of the partitioning. Kernighan and Lin proposed a bipartitioning procedure which starts with a random partitioning, and then repeatedly swaps a pair of vertices between the two parts, choosing the swap with the largest possible gain, irrespective of whether the gain is positive or negative; this allows the procedure to escape from local minima. Swapped vertices are then locked for the remaining part of the current round. The best partitioning encountered during a round is kept. Several rounds are carried out, each one starting from the best partitioning of the previous round. The result of the last round is taken as the final result. Fiduccia and Mattheyses [20] improved the speed of the procedure by using moves instead of swaps, and by using better data structures. The method is effective for a limited number of vertices, up to a few hundred. Beyond that range, it is best combined with a multilevel method to reduce the problem size.

For graph partitioning, many software packages are available: sequential partitioners Chaco [22], Metis [24], Scotch [32], Jostle [39], KaHIP [35], and parallel partitioners ParMetis [26] and PT-Scotch [14]. Graph partitioning is considerably faster than hypergraph partitioning and is often used for finite element meshes, where the edge cut is a reasonable approximation for the communication volume. For an extensive recent overview of graph partitioning, see [8]. Chris Walshaw maintains an online collection of graph partitioning problems, the Graph Partitioning Archive[2], with for each problem the best solution found so far, and links to the software that produced it, for imbalance values of $\varepsilon = 0, 0.01, 0.03, 0.05$.

For optimal graph partitioning, a large body of literature exists. Algorithms that provide an optimal solution to the problem are called *exact algorithms*. Often, such algorithms are based on the branch-and-bound approach [30], which organizes the search for an optimal solution as branches in a search tree, where each path from the root to a leaf represents a solution. The tree is searched in a depth-first fashion. Subtrees are pruned based on bounds for the solution: in case of a minimization, these are a lower bound LB on the best solution that can still be obtained in the current subtree, and an upper bound UB given by the best solution found so far. If $LB \geq UB$, the subtree cannot contain a solution better than the current best and hence it can safely be pruned.

Karisch, Rendl, and Clausen [23] solve graph bipartitioning problems to

[2] `http://staffweb.cms.gre.ac.uk/~wc06/partition/`

optimality with a branch-and-bound method based on a cutting plane approach that combines semidefinite and polyhedral relaxations. Their problem sizes are of the order 80–90 vertices for general graphs. For special graphs, e.g. deriving from 2D meshes, they solve larger problems. Sensen [36] solves the same problem using multicommodity flows. Felner [19] takes a purely combinatorial approach in his branch-and-bound algorithm for solving the graph bipartitioning problem with uniform edge weights. We take the same kind of approach in our sparse matrix bipartitioning and our algorithm has therefore certain similarities with Felner's. A major difference is in the minimization objective: we minimize communication volume instead of edge cut, and this leads to different lower bounds. The largest problem size achieved by Felner is 100 vertices and 1000 edges.

Hager, Phan, and Zhang [21] present an exact branch-and-bound algorithm for edge-weighted graph bipartitioning formulated as a continuous quadratic program with lower bounds based on semidefinite programming. They solve problems to optimality with a few hundred vertices in most cases; the largest problem solved, `KKT.capt09` has 2063 vertices and about 21,000 edges. Delling et al. [16] obtain optimal results by a purely combinatorial branch-and-bound algorithm based on packing-tree bounds and a graph contraction method, for various instances of the recent 10th DIMACS challenge on graph clustering and partitioning [1]. One of the larger instances solved was the open street map `luxembourg` with $114,599$ vertices and $119,666$ edges, which they solved in 38 s for $p = 2$ and $\varepsilon = 0$. The minimal edge cut for this problem is 17.

For optimal hypergraph partitioning and optimal sparse matrix partitioning, very little work has been done so far. Caldwell, Kahng, and Markov [9] develop two exact hypergraph partitioners for cell layout of electronic circuits, a branch-and-bound method and an enumerative method based on Gray codes, with branch-and-bound the better method, except for very small problems. The authors treat nets connecting two vertices (the 'graph part') in a special way, deriving a lower bound for inevitable cuts. They reached problem sizes of about 60 vertices, solved in 100 s. Kucar, Areibi, and Vannelli [29] survey different hypergraph partitioning techniques, including heuristics such as multilevel methods, simulated annealing, and genetic algorithms, and also an exact algorithm based on integer linear programming (ILP). Their largest problem solved to optimality has 1888 movable vertices, 1920 nets, and 5471 pins (corresponding to matrix nonzeros); it took 3 days to solve the problem. The multilevel hypergraph partitioner hMetis [25] was able to solve the same

problem in 0.33 s, and it managed to produce an optimal solution.

The Mondriaan package was used in a comparison with an ILP solver for an industrial problem [3] requiring partitioning of a software call graph (a directed graph) into modules with small interfaces. This was modeled using a hypergraph with the cut-net metric (cost 1 instead of a cost $\lambda_i - 1$ for a cut net). The largest problem solved was partitioning a COBOL program with 1100 subprograms and 2951 call edges into 8 modules, solved to optimality in 9 days.

To compute an optimal sparse matrix partitioning, the problem could be translated using the fine-grain model into a hypergraph with N vertices and $m + n$ nets, which could be solved by an exact hypergraph partitioning algorithm. The resulting hypergraph has a special structure, however, namely that every vertex is part of exactly two nets, one from a group of m nets (the row nets), and one from a group of n nets (the column nets). This is because every nonzero a_{ij} belongs to exactly one row i and one column j. Furthermore, the vertex weights and net costs are all 1. Thus, although an exact hypergraph partitioner could in principle be used to solve the problem optimally, an exact algorithm that exploits its special properties would do this much faster.

One way of exploiting the special properties is to avoid the translation to a hypergraph altogether, thus partitioning the matrix nonzeros themselves. This can be done optimally by a branch-and-bound algorithm that either directly partitions the nonzeros of the matrix in a straightforward manner, or considers the nonzeros of a matrix row (or column) together, and assigns them all to processor 0, or all to processor 1, or decides to cut the row (or column), thus incurring a communication, in which case the individual assignment of its nonzeros is irrelevant. This amounts to partitioning the matrix rows into three sets, and the same for the columns. The row/column-based approach is the most promising one and therefore we have chosen it as the basis of our exact algorithm, which will be explained in Section 4.2.

Kayaaslan et al. [27] present a heuristic method for hypergraph partitioning based on graph partitioning by vertex separator (GPVS), a procedure which for $p = 2$ partitions the vertices of a graph into two unconnected sets and a separator set. This is similar to our approach of partitioning the rows and columns into three sets, although we use it to obtain an optimal partitioning instead of a heuristic one. They apply GPVS to the net intersection graph (NIG) of the hypergraph. The result is a part for processor 0 and a part for processor 1. For the load balance, edges within part 0 or con-

nected to part 0 are counted towards processor 0, and similar for processor 1. Edges within the separator can be assigned arbitrarily, and correspond to *free* nonzeros (defined in Section 4.2) in the matrix; they only influence the load balance, but not the communication volume.

3 Medium-grain method

In this section, we briefly present the medium-grain method [34] for sparse matrix partitioning. In this method, the matrix A is first split by a simple procedure into two parts,

$$A = A^r + A^c, \tag{4}$$

where each nonzero of A is placed either in A^r or A^c. After that, a new $(m+n) \times (m+n)$ matrix B is formed,

$$B = \begin{bmatrix} I_n & (A^r)^T \\ A^c & I_m \end{bmatrix}, \tag{5}$$

where I_m is the identity matrix of size $m \times m$. Columns that only contain a diagonal nonzero (from I_m or I_n) are removed to prevent unnecessary communication. The resulting matrix is bipartitioned in 1D fashion by translating it according to the row-net model to a hypergraph with $m+n$ vertices corresponding to matrix columns, and $m+n$ nets corresponding to matrix rows, and then using a multilevel hypergraph bipartitioner, see Figure 2. In this partitioning, the weight of a column of B for the load-balance criterion should be taken as the number of original nonzeros it includes from A. The new nonzeros of the identity matrices I_m and I_n were created to represent the communication volume correctly, but they should not contribute to the column weight, as they do not count towards the original balance criterion of Eq. (2).

After the partitioning of B, the result is translated back into a partitioning of A, using the unique correspondence between off-diagonal nonzeros of B and nonzeros of A. In this method, nonzeros from a column of A that are placed in A^c are kept together during the partitioning, and the same holds for nonzeros from a row that are placed in A^r.

In our previous work [34], we have proven that the communication volume of the 1D partitioning of B is exactly the same as the volume of the corresponding partitioning of A. The proof is based on the connections which

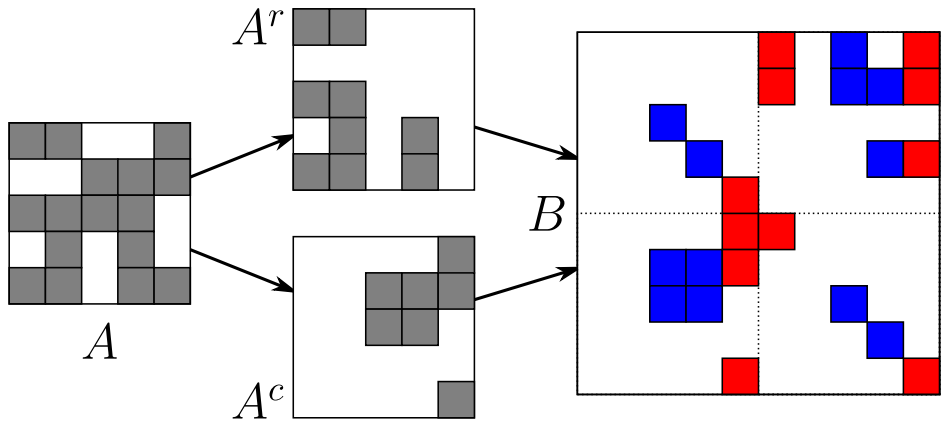


Figure 2: The medium-grain method applied to the matrix of Figure 1. The nonzeros of the sparse matrix A are split into parts A^r and A^c. Afterwards, the B matrix is formed and bipartitioned by column, indicated by color: red nonzeros are assigned to processor 0, blue nonzeros to processor 1. The corresponding bipartitioning of A is identical to the one shown in Figure 1 and Figure 3b.

the diagonal nonzeros of B establish between nonzeros from a row or column that were assigned to different parts A^r and A^c. More precisely, nonzero b_{ii} from the block I_n connects the nonzeros of column i of A (split among column i of the block A^c and row i of the block $(A^r)^T$); a similar connection is established by nonzeros from the block I_m.

For splitting A into A^r and A^c, a cheap and effective strategy is to place nonzero a_{ij} in A^c if column j has fewer nonzeros than row i, and in A^r otherwise. The motivation is that a column with fewer nonzeros has a better chance to stay together in a good partitioning. Ties are broken in a uniform manner by assigning them all to A^c or all to A^r, depending on the matrix dimensions, but they can also be broken in a more sophisticated manner, see [34], giving slightly better results. An exception to this placement strategy is the case where the nonzero a_{ij} is the only nonzero of its row, since such a row cannot be cut, so that it is better to place the nonzero in A^c, and similarly for the only nonzero in a column. Note that if we would take $A^c = A$ and $A^r = 0$, the medium-grain method reduces to the 1D row-net model (giving a column partitioning), since all columns are kept intact.

After executing a run of the medium-grain method, we have obtained a

bipartitioning of the nonzeros into two subsets A_0 and A_1, and we can use these subsets as the initialization for a new restricted run. This leads to an *iterative refinement* procedure, see [34]. The nonzeros of A_0 are entered into A^r, those of A_1 are entered into A^c, and the corresponding columns of B are assigned to processors 0 and 1, respectively. This ensures that the current volume equals that of the previous run of the medium-grain method. The new run is restricted in the sense that the coarsening and initial partitioning of the multilevel hypergraph bipartitioning are skipped, and only one level of refinement is carried out, namely the finest level, using the Kernighan–Lin/Fiduccia-Mattheyses algorithm [28, 20]. This implies that the volume can only decrease or stay the same. If no decrease is obtained, the roles of A^r and A^c are reversed, and so on, until no further decrease can be obtained in any direction. Iterative refinement is cheap, as it only involves the final part of a complete multilevel partitioning, and it is always worthwhile as a post-processing step, also in combination with other partitioners than the medium-grain method.

The time complexity of bipartitioning an $m \times n$ matrix A by the medium-grain method equals

$$T_{MG} = \mathcal{O}((m+n)C_{\max}^2), \tag{6}$$

where $C_{\max}$ is the maximum number of nonzeros in a single row or column of the matrix. The complexity can be determined similar to the analysis for the 1D column partitioning method given in Ref. [4, Section 12.7]. The main operation that determines the complexity is the coarsening of $m+n$ columns of the matrix B, computing their inner products with at most $C_{\max}$ other columns, accessing at most $C_{\max}$ nonzeros in every column. Splitting A into A^r and A^c and performing one iteration of iterative refinement are cheap, because both are linear in the number of nonzeros of A.

4 Branch-and-bound method

In this section, we describe the branch-and-bound method that we used to compute the optimal communication volume of bipartitionings of sparse matrices. First, we explain how branch-and-bound methods are able to find optimal solutions by discussing a straightforward method that finds optimal bipartitionings by partitioning the nonzeros directly. Then, in Section 4.2, we give the main contribution of this article: a branch-and-bound method

that still partitions the nonzeros, but only maintains the important information whether a row or column is assigned to processor 0, or to processor 1, or is cut. No partitioning information on individual nonzeros needs to be maintained, and this significantly reduces the computation time. In Section 4.3, we present three lower bounds on the communication volume of a partial solution that we use to decrease the number of feasible solutions to be considered. In Section 4.4, we present an additional lower bound that can be used as an alternative for one of the other bounds.

4.1 Directly bipartitioning the nonzeros

A straightforward way of finding the optimal bipartitioning of a matrix A is to simply try all possible bipartitionings of the nonzeros of A into two sets A_0 and A_1 that obey Eq. (2), and return the bipartitioning with the lowest communication volume. One method of traversing all possible bipartitionings in an efficient way is the branching method. Here, we traverse all bipartitionings by generating *partial* bipartitionings $\hat{A}_0$ and $\hat{A}_1$ (denoted by a hat), where only a subset of all nonzeros of A have been partitioned. By recursively adding and removing nonzeros to and from the partial bipartitionings, all 2^N different bipartitionings can be generated. Bipartitionings that do not obey Eq. (2) can be skipped during the branching method.

At any point during the branching method, an upper bound UB to the optimal communication volume is available by returning the lowest communication volume encountered so far for a complete solution. The main idea of the branch-and-bound method is to use this upper bound to skip entire parts of the solution space during branching. To do so, one needs a lower bound $LB(\hat{A}_0, \hat{A}_1)$ on the communication volume of all full bipartitionings that can be found by extending the current partial bipartitioning $\hat{A}_0$ and $\hat{A}_1$. For example, the number of rows and columns already cut by the partial bipartitioning is such a lower bound, since in any extension of $\hat{A}_0$ and $\hat{A}_1$, those rows and columns will be cut as well. If, during the branching method, the lower bound $LB(\hat{A}_0, \hat{A}_1)$ of the current partial bipartitioning is higher than or equal to the current upper bound UB, we can safely skip all solutions that are an extension of $\hat{A}_0$ and $\hat{A}_1$, since we know that these will not have a lower communication volume than the current upper bound. By doing so, the number of bipartitionings to be traversed to find the optimal one can be greatly reduced.

The main problem with the approach of generating all possible biparti-

tionings by recursively adding and removing nonzeros to and from $\hat{A}_0$ and $\hat{A}_1$ is that it is difficult to find good lower bounds on the communication volume when given a partial bipartitioning. For example, when a row or column is already cut in a partial bipartitioning of the nonzeros, all unassigned nonzeros in that row or column can be assigned arbitrarily without increasing the communication volume in that row or column. The lack of a good lower bound for partial solutions makes it difficult to bound the number of solutions to be considered in the branch-and-bound method.

4.2 Partitioning the rows and columns into three sets

Instead of directly partitioning the nonzeros into two sets, we can also partition the rows and columns of the matrix into three sets. In this case, a row or column can be completely assigned to processor 0, completely assigned to processor 1, or cut. In a completely assigned row or column, all nonzeros in that row or column are assigned to the same processor. In a cut row or column, a subset of the nonzeros in that row or column is assigned to one processor, and the remaining nonzeros are assigned to the other processor. For the communication volume, it does not matter which nonzeros exactly are assigned to which processor. A row can only be assigned to a processor if this does not conflict with an earlier column assignment: in a valid partitioning, a nonzero a_{ij} cannot reside both in a row assigned to processor 0 and a column assigned to processor 1, or vice versa. A similar rule applies when assigning columns to processors. Rows and columns can always be assigned as being cut. For reasons of symmetry, the first row or column to be completely assigned to a processor can always be assigned to processor 0.

Formally, we partition the set of rows and columns of the matrix into three sets B_0, B_1, and B_c, which represent the rows and columns that are completely assigned to processor 0, completely assigned to processor 1, and cut, respectively. Similarly to $\hat{A}_0$ and $\hat{A}_1$, we can traverse all possible partitionings by recursively generating partial partitionings $\hat{B}_0$, $\hat{B}_1$, and $\hat{B}_c$. Given a full partitioning, we can find a corresponding partitioning of the matrix nonzeros into three sets D_0, D_1, and D_{free}, by:

$$\begin{aligned} D_0 &= \{a_{ij} \in A : \text{ row } i \in B_0 \text{ or column } j \in B_0\}, \\ D_1 &= \{a_{ij} \in A : \text{ row } i \in B_1 \text{ or column } j \in B_1\}, \\ D_{free} &= \{a_{ij} \in A : \text{ row } i \in B_c \text{ and column } j \in B_c\}. \end{aligned} \tag{7}$$

This partitioning of the matrix nonzeros can be translated to a matrix bipartitioning by assigning all nonzeros in D_0 to processor 0 and assigning all nonzeros in D_1 to processor 1. Note that a nonzero cannot be in both D_0 and D_1, since that will result in an invalid partitioning of the matrix nonzeros. This fact greatly reduces the number of partitionings to be considered in the branch-and-bound method. The nonzeros in D_{free} can be assigned freely without increasing the communication volume of the resulting bipartitioning, since both the rows and columns of these nonzeros are cut, by construction. Note that assignment of the free nonzeros in such a way that it would reduce the communication volume, e.g. all nonzeros in a row being free and assigned to processor 0, is already explored as a different solution in the branching tree, in this case with assignment of the row to B_0 instead of B_c.

Given a partitioning of the rows and columns of the matrix, the communication volume of the corresponding bipartitioning can be found by counting the number of rows and columns assigned to B_c:

$$Vol(B_0, B_1, B_c) = |B_c| \,. \tag{8}$$

Furthermore, to ensure that the load imbalance constraint is satisfied, we only need to count nonzeros in rows and columns that are completely assigned to a single processor, since the nonzeros that are both in a cut row and a cut column can be assigned freely:

$$|D_i| \leq (1+\varepsilon) \left\lceil \frac{N}{2} \right\rceil, \quad \text{for } i \in \{0,1\}. \tag{9}$$

Finally, note that it is also possible to find optimal 1D bipartitionings using the same branch-and-bound approach by only allowing either rows or columns to be assigned to B_c.

At first, the proposed method of partitioning the rows and columns does not seem to be an improvement over partitioning the nonzeros: in the new method, a maximum of 3^{m+n} solutions have to be considered, compared to a maximum of 2^N solutions in the previous method. Many partitionings of the proposed method, however, represent *invalid* solutions. If the matrix element a_{ij} is nonzero, and row i is in B_0 while column j is in B_1 (or vice versa), the resulting branch does not represent a valid matrix bipartitioning. Furthermore, the new branching method allows for better lower bounds on a partial solution compared to the straightforward method, which will be explained in Section 4.3. Because of the number of invalid solutions and

the better lower bounds, a larger part of solution space can be skipped in the branch-and-bound method when partitioning the rows and columns instead of the nonzeros of a matrix. The result is that in the new method, a significantly lower number of partitionings has to be traversed than in the straightforward method.

4.3 Lower bounds on the communication volume

Given partial partitionings $\hat{B}_0$, $\hat{B}_1$, and $\hat{B}_c$, we would like to have a good lower bound $LB(\hat{B}_0, \hat{B}_1, \hat{B}_c)$ to be able to skip large parts of the solution space in the branch-and-bound method. In this article, we use three independent lower bounds, with the final bound being the sum of the three:

$$\begin{aligned} LB(\hat{B}_0, \hat{B}_1, \hat{B}_c) =& L_1(\hat{B}_0, \hat{B}_1, \hat{B}_c) + L_2(\hat{B}_0, \hat{B}_1, \hat{B}_c) \\ &+ L_3(\hat{B}_0, \hat{B}_1, \hat{B}_c). \end{aligned} \tag{10}$$

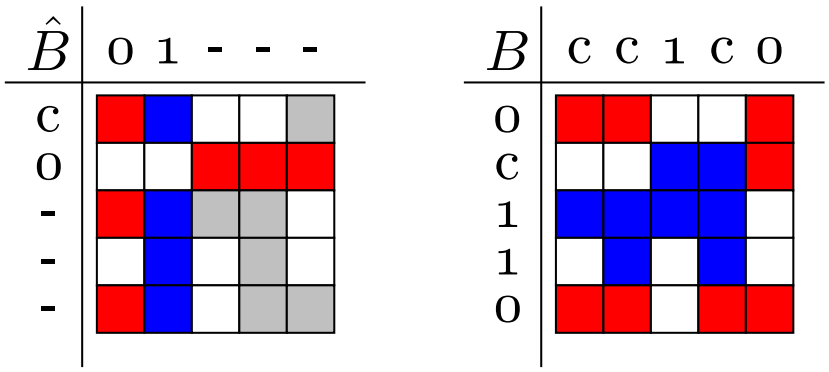
To explain the different lower bounds, we use an example of a partial partitioning of a small matrix, shown in Figure 3a.

The first lower bound is the number of rows and columns already *explicitly cut* in the current partial partitioning by being included in $\hat{B}_c$. This lower bound can simply be computed by counting the number of elements in $\hat{B}_c$:

$$L_1(\hat{B}_0, \hat{B}_1, \hat{B}_c) = \left|\hat{B}_c\right|. \tag{11}$$

Since the communication volume of a full partitioning is equal to the number of elements in B_c (Eq. (8)), and any extension of $\hat{B}_c$ includes all elements of $\hat{B}_c$ as well, we see that $L_1(\hat{B}_0, \hat{B}_1, \hat{B}_c)$ is indeed a lower bound on the communication volume. For the example of Figure 3a, only the first row is included in $\hat{B}_c$, so $L_1(\hat{B}_0, \hat{B}_1, \hat{B}_c) = 1$.

The second lower bound is based on *implicitly* cut rows and columns: those that are currently unassigned, but have to be assigned to B_c in the future to maintain a valid solution. For example, if row i is assigned to $\hat{B}_0$ and row i' to $\hat{B}_1$, any unassigned column that has a nonzero in both row i and i' has to be assigned to B_c in the future, since assigning such a column to B_0 or B_1 will result in an invalid partitioning. A similar reasoning can be applied to unassigned rows instead of columns. In the example of Figure 3a, both the third row and fifth row are implicitly cut by the assignment of the first column to $\hat{B}_0$ and the second column to $\hat{B}_1$. Therefore, $L_2(\hat{B}_0, \hat{B}_1, \hat{B}_c) = 2$ in the example.



(a) (b)

Figure 3: A partial partitioning (a) and optimal partitioning (b) of the rows and columns of a 5×5 matrix with 16 nonzeros, with perfect load balance. Next to each row and column, the subset to which that row or column is assigned is indicated by 0, 1, and c for $\hat{B}_0$, $\hat{B}_1$, and $\hat{B}_c$, respectively. Unassigned rows and columns are indicated by $-$. The corresponding (partial) bipartitioning of the matrix nonzeros is indicated by color: red nonzeros are assigned to processor 0, blue nonzeros to processor 1, and gray nonzeros are unassigned. The optimal communication volume is 4.

The third lower bound is based on rows and columns that are partially assigned. A currently unassigned row is *partially assigned* if it has at least one nonzero in a column that is assigned to $\hat{B}_0$, but no nonzeros that are in columns assigned to $\hat{B}_1$, or vice-versa. A similar definition can be given for partially assigned columns instead of rows. In the example of Figure 3a, the fourth row is partially assigned to $\hat{B}_1$, and the third, fourth, and fifth columns are all partially assigned to $\hat{B}_0$. Now, in order to avoid cutting a row or column that is partially assigned to $\hat{B}_0$ in the future, all unassigned nonzeros in that row or column have to be assigned to processor 0 as well. We can count all nonzeros that have to be assigned to processor 0 to avoid communication in rows and columns that are partially assigned to $\hat{B}_0$, giving a total number s_0. Similarly, we can calculate the number s_1 for processor 1.

If s_0 exceeds the number of nonzeros that can still be assigned to processor 0 due to the load balance constraint of Eq. (2), at least some of the partially assigned rows and columns have to be cut in the future. To cut the least number of rows and columns, it is best to cut them in order of decreasing number of unassigned nonzeros in them, until we are left with a number of

unassigned nonzeros that obeys the load imbalance constraint. The smallest number of rows and columns to be cut in this way can be used as a lower bound on the communication volume. Note that it is possible that a nonzero is in both a row and column that is partially assigned to processor 0. During calculation of L_3, we might cut both the row and column of that nonzero, and subtract the nonzero twice from the count. Since we stop cutting rows and columns when the current nonzero count is small enough, this can only lead to an underestimation of the number of rows and columns to be cut, and therefore, the resulting value can still be used as a lower bound. A similar but independent lower bound can be calculated for processor 1 instead of processor 0, and the final bound L_3 is the sum of both bounds. To justify addition of the two bounds, note that rows (or columns) cannot be partially assigned to both processor 0 and processor 1, because in that case they would be implicitly cut.

To clarify the third lower bound, we calculate its value for the example of Figure 3a. If we aim to have perfect load balance (eight nonzeros assigned to each processor), only two more nonzeros can be assigned to processor 0, since six have already been assigned to it. Now, the third, fourth, and fifth columns are partially assigned to processor 0, and contain six unassigned nonzeros, so $s_0 = 6$. The six unassigned nonzeros cannot all be assigned to processor 0 because of the load balance constraint, so some of the partially assigned columns have to be cut. As explained above, it is best to cut them in decreasing number of unassigned nonzeros, so we start by cutting the fourth column. We are left with three unassigned nonzeros in the third and fifth columns, which is still more than the two nonzeros that can be assigned to processor 0. Therefore, the fifth column has to be cut as well, and we know that at least two of the partially assigned columns have to be cut to obey the load balance constraint, so $L_3(\hat{B}_0, \hat{B}_1, \hat{B}_c)$ is at least two. Since $s_1 = 1$ is smaller than four, the number of nonzeros that can still be assigned to processor 1, we know that processor 1 does not increase the third lower bound further. Therefore, in the example of Figure 3a, $L_3(\hat{B}_0, \hat{B}_1, \hat{B}_c) = 2$.

For the example of Figure 3a, the sum of the three lower bounds is equal to $LB(\hat{B}_0, \hat{B}_1, \hat{B}_c) = 1+2+2 = 5$. The optimal bipartitioning, shown in Figure 3b, has a communication volume of 4. Therefore, if we had already found an optimal bipartitioning during the branch-and-bound method, we would be able to skip all partitionings that are an extension of the one shown in Figure 3a, since we know that the communication volume of these partitionings will not be smaller than the currently best known communication volume.

To summarize, the first lower bound is equal to the number of rows and columns that are cut by inclusion in $\hat{B}_c$, the second lower bound is equal to the number of rows and columns implicitly cut by assignments of other columns and rows, respectively, and the third bound is based on partially assigned rows and columns and the load balance constraint. In order to maximize the values of the different lower bounds, it was experimentally found to be beneficial to assign the rows and columns of the matrix in decreasing order of nonzeros in them. Since the lower bounds have to be calculated for every partial partitioning that is evaluated during the branch-and-bound method, the time it takes to compute them has a significant impact on the total computation time of the entire method. In [33], it was proven that by careful accounting of additional variables during the branching procedure, the three bounds can be calculated in $\mathcal{O}(1)$, $\mathcal{O}(C_{\max})$, and $\mathcal{O}(C_{\max})$ time, respectively.

The proof that the three bounds can be added into a lower bound $L_1 + L_2 + L_3$ follows from considering the rows and columns that correspond to each bound, as illustrated by the block structure of the permuted matrix in Figure 4. Rows and columns have been assigned to three blocks $\hat{B}_0, \hat{B}_1, \hat{B}_c$, and this generates four additional row blocks, namely rows partially assigned to processor 0 (P_0), partially assigned to 1 (P_1), implicitly cut rows (I_c), and unassigned rows; similarly, this generates four additional column blocks. Rows and columns in $\hat{B}_c$ correspond to L_1, those in I_c to L_2, those in P_0 and P_1 to L_3. Since a row can be part of at most one block $\hat{B}_c, I_c, P_0, P_1$, it can only be counted in one of the three bounds. The same holds for columns.

4.4 Dynamic maximum-cardinality bipartite graph matching

In this section, we present an additional bound $L_4(\hat{B}_0, \hat{B}_1, \hat{B}_c)$ which can be used as an alternative for L_3, based on a different way of using partially assigned rows and columns. We can use this bound by replacing L_3 in Eq. (10) with $\max(L_3, L_4)$. The bound is caused by nonzeros where partial assignments conflict: in Figure 4, the nonzero in the row from row block P_1 and the middle column from column block P_0 implies that either the row or the column must be cut, giving a new bound $L_4 = 1$ for this example. In general, the bound L_4 is created by two independent conflict submatrices, one submatrix consisting of rows from row block P_0 and columns from column block P_1, and the other with the roles reversed.

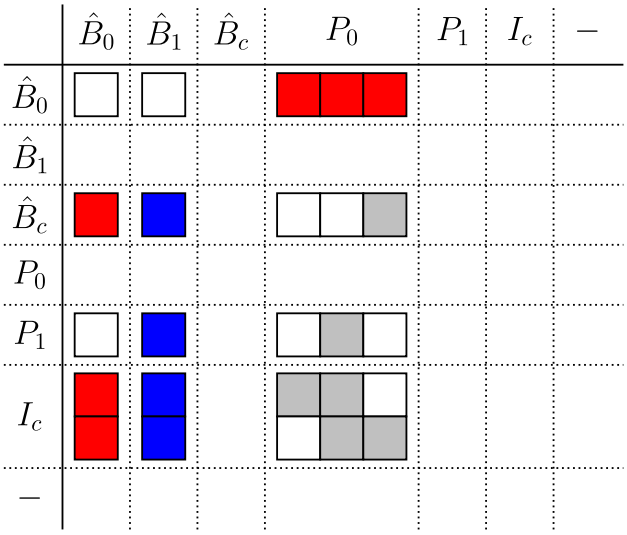


Figure 4: Block structure of the matrix of Figure 3a obtained by permuting rows into contiguous blocks of rows assigned, respectively, to processor 0, to 1, cut rows, rows partially assigned to 0, to 1, implicitly cut rows, and unassigned rows, and similarly for columns.

A conflict submatrix can have more than one nonzero. In this case, the lower bound can be obtained by solving a matching problem for a bipartite graph, as follows. Let us define A' as the submatrix of A that contains all matrix elements in the intersection of row block P_0 and column block P_1. If we choose a subset M of nonzeros from A' with at most one nonzero in every row and at most one in every column, then each nonzero will lead to either a cut row or a cut column, and the number of nonzeros in M provides a lower bound L_4 on the communication volume caused by M.

We can translate the choice of M into a bipartite graph matching problem, by defining the vertex set V_0 corresponding to the rows i partially assigned to processor 0, and the vertex set V_1 corresponding to the columns j partially assigned to processor 1, with an edge $(i, j) \in E$ if and only if a_{ij} is nonzero. The chosen subset M, now viewed as a set of edges, has at most one edge connected to a single vertex, and thus corresponds to a *matching* $M \subseteq E$. The lower bound equals the cardinality of M, i.e., $L_4 = |M|$. Finding the largest lower bound therefore means finding a maximum matching in the bipartite graph, i.e., a matching with the largest cardinality $|M|$. Since the

problem is relatively small, we can attempt to solve it exactly, using so-called augmenting paths.

An *augmenting path* for a matching M in a graph G is a simple path (no cycles) $i_0, i_1, \ldots, i_k$, of odd length k, where the start vertex i_0 and the end vertex i_k are unmatched, and $(i_r, i_{r+1}) \in M$ for odd r and $(i_r, i_{r+1}) \in E \backslash M$ for even r. If an augmenting path exists, a matching with one extra match can be constructed by flipping the matches along the path, thus including edges (i_r, i_{r+1}) in M for even r and excluding them for odd r. Berge's theorem [2] states that a matching is maximum if and only if no augmenting path exists.

In our branch-and-bound algorithm, the matrix A' grows and shrinks by adding or removing partially assigned rows and columns. In terms of the bipartite graph, adding a row (or column) with a set of nonzeros means adding a vertex with a set of its edges, and removing a row (or column) means deleting a vertex with all its edges. During all these operations we maintain a maximum matching, and we do this by the following dynamic matching algorithm with vertex updates. This dynamic algorithm is based on straightforward application of Berge's theorem, and on the well-known Hungarian algorithm for (static) maximum-cardinality bipartite graph matching. We start with an empty matrix A' and hence an empty edge set E, which has the trivial maximum matching $M = \emptyset$.

Assume we add a vertex i and a set of edges (i, j) to the graph $G = (V_0 \cup V_1, E)$, giving a new graph G'. Let us first consider the case where there exists an augmenting path starting in i in the new graph for the existing matching M. Flipping the edges creates a new matching M' with $|M'| = |M| + 1$. It is easy to see that M' is a maximum matching in G'. Proof: if M' is not a maximum matching, there would exist a matching M'' with cardinality $|M''| \geq |M| + 2$. If i is matched in M'', we can delete its matched edge (i, j) from M'' and then obtain a matching in G with cardinality at least $|M| + 1$, which is a contradiction, since M is a maximum matching. If i is not matched in M'', then M'' is a matching in G, also leading to a contradiction.

Let us now consider the case where no augmenting path starting in i exists in G' for M. In that case, we claim that M is a maximum matching in G'. Proof: if the claim is false, there would exist an augmenting path in G' by Berge's theorem. This path cannot contain the unmatched vertex i: i cannot be an interior vertex (since all interior vertices are matched), nor a starting or end vertex. Therefore, this path is also an augmenting path in G, contradicting the fact that M is maximum in G.

As a result, a maximum matching can be maintained when adding a new

vertex i by looking for an augmenting path starting at i and flipping the edges of the path if it exists. For a bipartite graph, an augmenting path can be found by building a Breadth-First Search (BFS) tree containing all alternating paths (with edges alternatingly inside and outside M) starting from vertex i. For more details, see e.g. the book by Bondy and Murty [7, Chap. 16]. The construction of the tree can be terminated once an unmatched vertex is reached.

A maximum matching can also be maintained when deleting a vertex i and its edges. If i is unmatched, the matching is not changed and remains maximum. If i is matched to a vertex j, we look for an augmenting path starting at j; note that j becomes unmatched in the new matching $M' = M \backslash \{(i, j)\}$. If such an augmenting path exists, flipping its edges gives a new matching M'' with $|M''| = |M|$, which is easily seen to be a maximum matching in G'. If no such augmenting path exists, M' is a maximum matching. Proof: if M' is not a maximum matching, there would be an augmenting path for M' not containing j. This would be an augmenting path for M as well, giving a contradiction.

The fourth bound can be calculated with $\mathcal{O}(C_{\max})$ vertex additions or deletions, each of which requires $\mathcal{O}(|E|)$ operations, equal to the number of nonzeros in the conflict block. In practice, there will be far fewer additions or deletions: for instance, an addition only occurs if there is a new partial assignment, and an augmenting path only has to be searched for if there is a new conflicting nonzero.

5 Experiments

We implemented the proposed branch-and-bound method in the C programming language, using the library of the Mondriaan software package, version 4.0. The software will be made available with an open-source license at the website of the Mondriaan software. The program was compiled using the GNU Compiler Collection, version 4.8.3, and executed on an Intel Core i7-2600K 3.4 GHz processor with 16 GB of RAM, under the Fedora 20 Linux operating system (Linux kernel 3.78.8, x86_64).

To test the branch-and-bound method, we attempted to find optimal communication volumes for bipartitionings of all matrices of the University of Florida sparse matrix collection [15] with at most 1000 nonzeros, of which there are 217. For each matrix, we tried to find the optimal communication

volume for $\varepsilon = 0.03$, a value commonly used in experiments of previous studies [4, 10, 38]. Since the problem of matrix bipartitioning is NP-hard, the branch-and-bound method may fail to find the optimal communication volume in a reasonable time for some matrices. Therefore, we stop the method after one day of computation, and report a failure for that matrix. Note that in these cases, we would still obtain an upper bound to the optimal volume (the lowest communication volume of any solution that was found), but it is not guaranteed to be optimal.

The computation time of the branch-and-bound method can be greatly decreased by starting with a good upper bound on the optimal communication volume. Of course, the communication volume of a feasible bipartitioning obtained by a heuristic method can be used as such an upper bound. In the experiments performed in this article, we used the medium-grain method to obtain an initial upper bound before starting the branch-and-bound method. The medium-grain bipartitioning was performed by Mondriaan 4.0 with its default options. During the algorithm, the current upper bound UB is decreased whenever a better complete solution than the current best is found. The initial lower bound on the communication volume is 0; the current lower bound LB used to prune the tree in case $LB \geq UB$ depends on the current partial solution.

When bipartitioning with $\varepsilon = 0.03$, the optimal communication volume was found within a single day of computation for 85% of the matrices. In Figure 5, the fraction of matrices that was optimally bipartitioned is shown as a function of computation time. Note that for 58% of the matrices, an optimal bipartitioning was found even within a second of computation. There are only a few matrices for which the optimal volume was found in a time between a minute and a day. This suggests that matrices can be divided into two groups: the easy ones that are solvable within a reasonable time (i.e. within a minute), and those that are difficult. In Figure 6, the fraction of matrices that was optimally bipartitioned is shown as a function of the number of matrix nonzeros. The results show that up to about 250 nonzeros, all matrices are solvable within a single day of computation. For higher numbers of nonzeros, the fraction gradually decreases to 85% at $N = 1000$.

In Figure 7, optimal bipartitionings are given for selected matrices, for $\varepsilon = 0.03$. Note that, for some of the matrices, the number of nonzeros in D_{free} enables achieving perfect load balance with the same volume. The observation that the optimal bipartitioning of many matrices contains free nonzeros can also be applied in heuristic methods: given a heuristic solution, it may

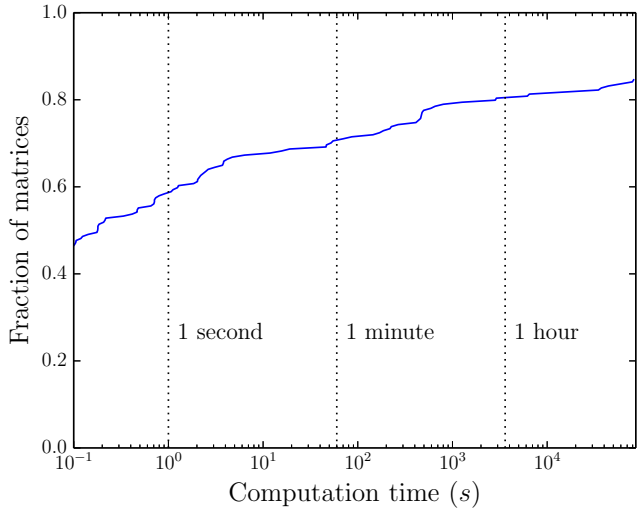


Figure 5: The fraction of matrices with at most 1000 nonzeros for which the optimal communication volume was found within the time indicated by the x-axis, with a maximum time of one day.

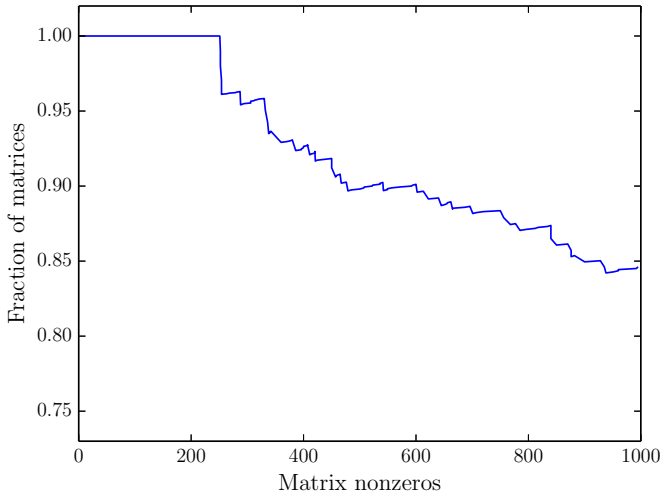


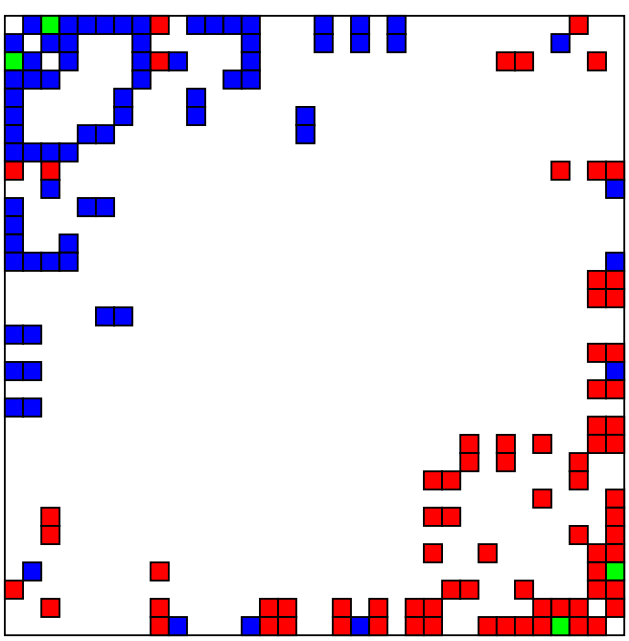
Figure 6: The fraction of matrices with a number of nonzeros smaller than or equal to the x-axis value for which the optimal communication volume was found within a single day of computation.

Matrix	m	n	N	Vol_{Loc}	Vol_{MG}	Vol_{FG}	Vol_{opt}	Time (s)
karate	34	34	156	23.01 ± 0.10	9.69 ± 0.46	8.71 ± 0.64	8	0.00
divorce	50	9	225	9.00 ± 0.00	9.00 ± 0.00	9.00 ± 0.00	8	0.00
cage5	37	37	233	28.27 ± 1.71	15.41 ± 2.11	14.53 ± 0.92	14	0.21
Sandi_authors	86	86	248	10.84 ± 0.80	5.28 ± 0.99	4.56 ± 0.84	4	0.00
mesh1e1	48	48	306	21.13 ± 1.43	20.04 ± 1.07	18.08 ± 0.37	18	463.33
impcol_b	59	59	312	15.17 ± 3.82	15.75 ± 4.36	12.22 ± 1.47	10	412.51
chesapeake	39	39	340	35.11 ± 0.31	18.61 ± 0.84	18.90 ± 1.26	16	0.19
steam3	80	80	928	12.52 ± 6.19	8.45 ± 2.18	8.32 ± 1.57	8	179.83

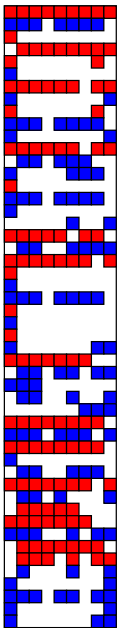
Table 1: The dimensions of the matrices of Figure 7, with the mean communication volume and standard deviation of 100 runs of the localbest method (Vol_{Loc}), medium-grain method (Vol_{MG}), and fine-grain method (Vol_{FG}), computed by the Mondriaan 4.0 software with default options. Also given are the optimal communication volume (Vol_{opt}) and computation time of the branch-and-bound method.

be possible to improve its load balance by redistributing its free nonzeros. This idea can be extended to partitioning for more than two processors as well, in which case each nonzero would have a set of processors to which it can be freely assigned without increasing the communication volume.

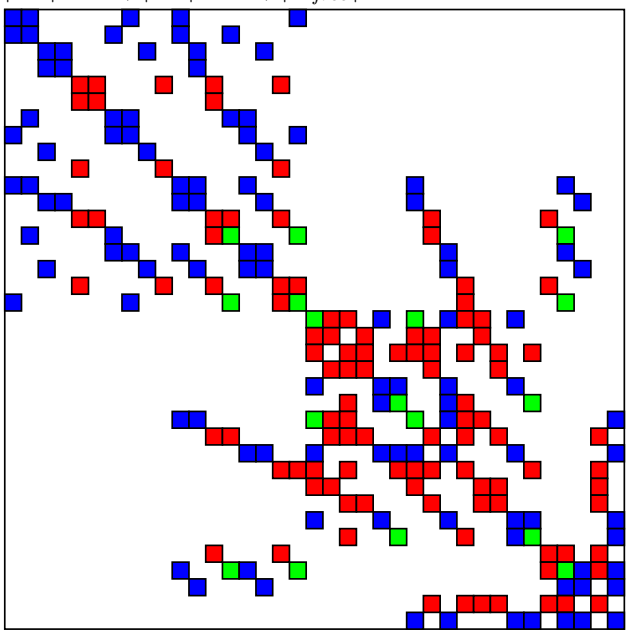
As explained before, optimal bipartitionings can be helpful in inspiring new ideas for improving heuristic solution methods. For example, the optimal bipartitioning of the divorce matrix (Figure 7b) suggests that for very rectangular matrices, it is best to look for a 1D partitioning in the direction of the smallest dimension, as was also experimentally found in [12]. Generally, Figure 7 shows that optimal bipartitionings are usually 2D. Furthermore, rows and columns with a relatively large number of nonzeros tend to be cut, while rows and columns with relatively few nonzeros are often assigned to a single processor. These two considerations inspired us to design the medium-grain method, which can produce 2D bipartitionings, and in which the initial split algorithm ensures that rows and columns with only relatively few nonzeros are assigned to a single processor, while rows and columns with a relatively large number of nonzeros are allowed to be cut. For the matrices shown in Figure 7, the optimal communication volume and time it took to compute them is given in Table 1, along with the mean communication volume of 100 runs of the localbest method, medium-grain method, and fine-grain method.



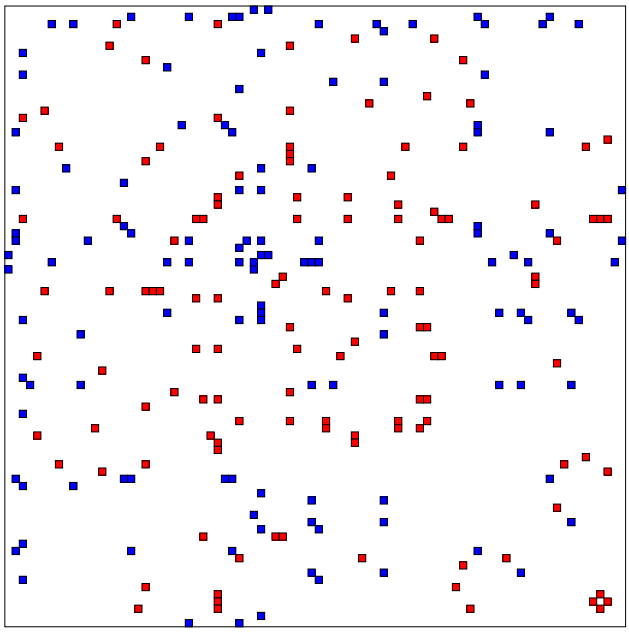
(a) `karate`, 34×34, 156 nonzeros, $Vol_{opt} = 8$, $|D_0| = 78$, $|D_1| = 74$, $|D_{free}| = 4$



(b) `divorce`, 50×9, 225 nonzeros, $Vol_{opt} = 8$, $|D_0| = 113$, $|D_1| = 112$, $|D_{free}| = 0$

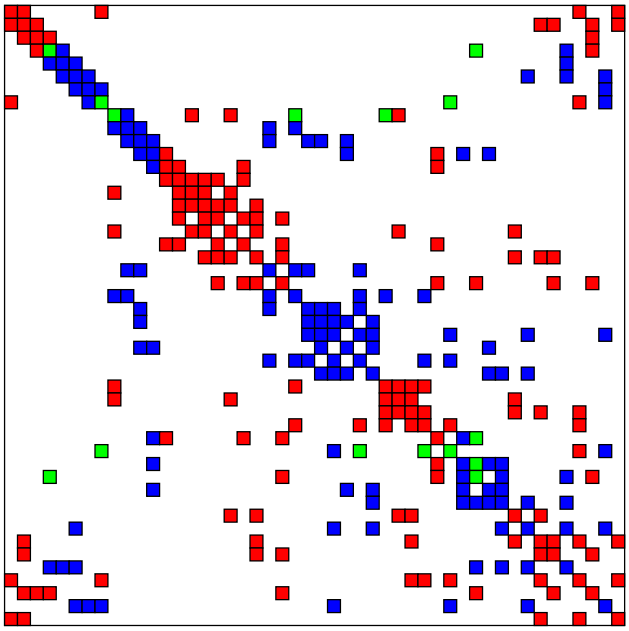


(c) `cage5`, 37×37, 233 nonzeros, $Vol_{opt} = 14$, $|D_0| = 106$, $|D_1| = 110$, $|D_{free}| = 17$

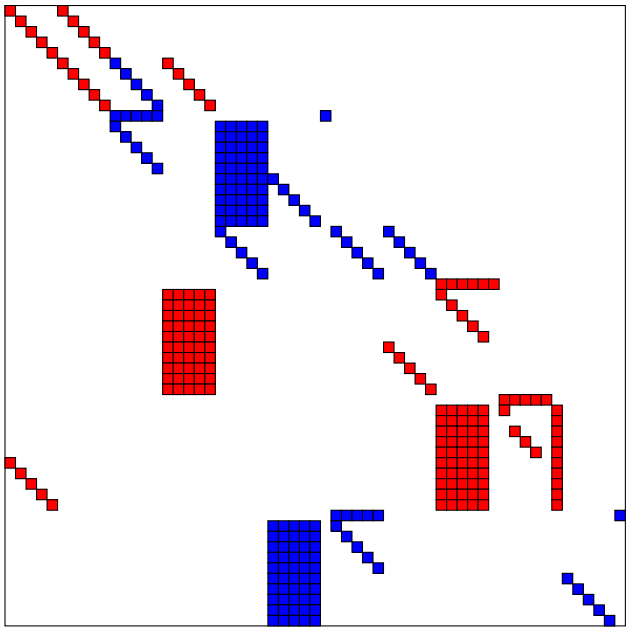


(d) `Sandi_authors`, 86×86, 248 nonzeros, $Vol_{opt} = 4$, $|D_0| = 124$, $|D_1| = 124$, $|D_{free}| = 0$

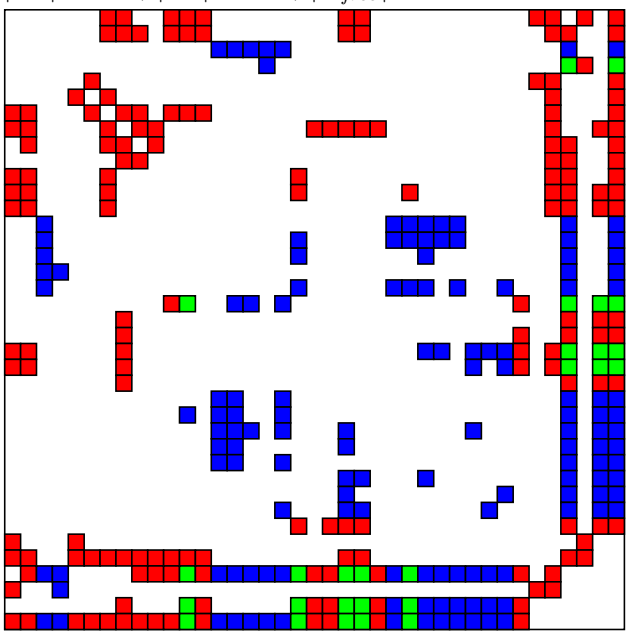
Figure 7: Optimal bipartitionings for $\varepsilon = 0.03$, calculated by the proposed branch-and-bound method. Nonzeros assigned to processor 0 are shown in red and nonzeros assigned to processor 1 are shown in blue. Nonzeros that can be freely assigned without increasing the communication volume (i.e. those that are assigned to D_{free}) are shown in green. Note that not all assignments of the nonzeros in D_{free} have to obey the load balance constraint, but we are guaranteed that at least one assignment exists that obeys it.



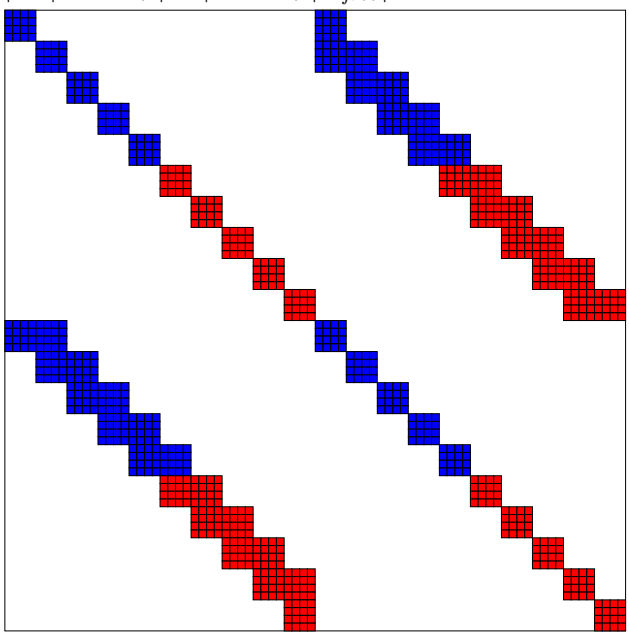
(e) `mesh1e1`, 48×48, 306 nonzeros, $Vol_{opt} = 18$, $|D_0| = 156$, $|D_1| = 135$, $|D_{free}| = 15$



(f) `impcol_b`, 59×59, 312 nonzeros, $Vol_{opt} = 10$, $|D_0| = 160$, $|D_1| = 152$, $|D_{free}| = 0$



(g) `chesapeake`, 39×39, 340 nonzeros, $Vol_{opt} = 16$, $|D_0| = 172$, $|D_1| = 141$, $|D_{free}| = 27$



(h) `steam3`, 80×80, 928 nonzeros, $Vol_{opt} = 8$, $|D_0| = 464$, $|D_1| = 464$, $|D_{free}| = 0$

Figure 7: (Continued)

Matrix	m	n	N	Vol_{Loc}	Vol_{MG}	Vol_{FG}	Vol_{opt}	Time (s)
stoch_aircraft	3754	7517	20267	14 ± 2	14 ± 3	13 ± 0	6	0.39
rosen1	520	1544	23794	8 ± 0	8 ± 0	24 ± 17	8	0.03
add32	4960	4960	23884	40 ± 11	13 ± 5	13 ± 5	4	381.29
mhd4800b	4800	4800	27520	3 ± 0	2 ± 0	2 ± 0	2	161.83
Chebyshev3	4101	4101	36879	4 ± 0	22 ± 7	15 ± 4	4	0.07
rosen2	1032	3080	47536	8 ± 0	8 ± 0	33 ± 21	8	0.05
lp_fit2p	3000	13525	50284	25 ± 0	25 ± 0	70 ± 11	21	0.79
rosen10	2056	6152	64192	8 ± 0	8 ± 0	26 ± 11	8	0.10
c-30	5321	5321	65693	1583 ± 151	43 ± 7	790 ± 21	30	6.07
lp_fit2d	25	10524	129042	25 ± 0	25 ± 0	27 ± 1	21	0.76

Table 2: Dimensions and computed communication volumes of the ten largest matrices solved to optimality within one hour of computation. For explanation of the abbreviations, see Table 1.

For some matrices with significantly more nonzeros than 1000, the optimal communication volume can still be found within reasonable time. In Table 2, the ten largest matrices for which an optimal volume was found within an hour of computation are shown. Note that these matrices generally have a very specific structure, which enables us to find the optimal volume. However, the results show that the matrices are not trivially solvable: the heuristic bipartitioners are not always able to get close to the optimal volume. The c-30 matrix is interesting in this regard, since the localbest and fine-grain method produce bipartitionings with a significantly larger communication volume compared to the optimal volume. A detailed investigation of the c-30 matrix might yield improvements to the existing heuristic methods.

To compare the performance of heuristic solvers with the optimal communication volume in more detail, we use performance profiles, which were introduced by Dolan and Moré [18] as a tool to compare different methods for a certain metric over a large test set. In this case, a performance profile shows, for each heuristic method, the fraction of matrices for which the mean communication volume of 100 runs is within some factor of the optimal communication volume. For example, if the performance profile of a method shows a fraction of 0.9 at factor 2, it shows that for 90% of the matrices, the mean communication volume of 100 runs of that method was less than or equal to two times the optimal communication volume. Matrices for which

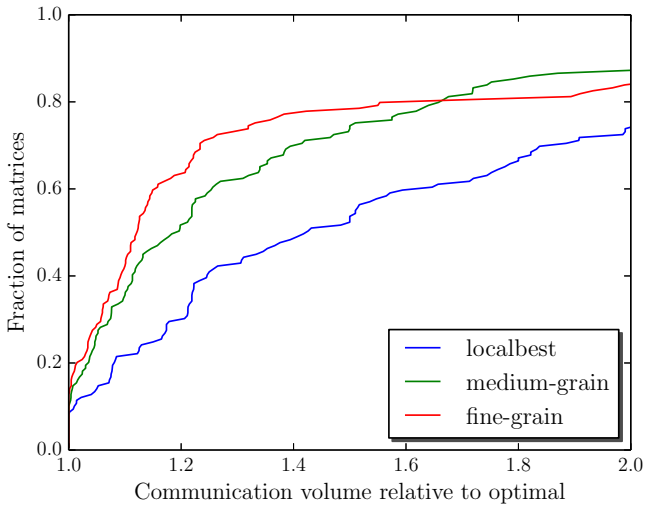


Figure 8: Performance profile plot comparing the optimal communication volume of all matrices with up to 1000 nonzeros with the mean communication volume of 100 runs of the localbest method, medium-grain method, and fine-grain method, computed by the Mondriaan 4.0 software with default options.

an optimal volume was not found within a single day of computation and those with an optimal communication volume of zero were removed from the set, since they cannot be represented in the performance profile.

In Figure 8, the performance profile plot is shown, for all matrices with up to 1000 nonzeros, for the localbest method, the medium-grain method with iterative refinement, and the fine-grain method. The results show that the 1D localbest method generally produces bipartitionings with a significantly higher communication volume than the optimal volume. Furthermore, for these small matrices, the fine-grain method is slightly better than the medium-grain method. For about 20% of the matrices, each heuristic produces, on average, bipartitionings with at least two times the optimal communication volume. This suggests that there is room for improvement of the different heuristics, or their implementation in Mondriaan 4.0.

6 Conclusions and future work

In this article, an exact branch-and-bound algorithm for sparse matrix bipartitioning was introduced. Given a matrix and allowed load imbalance, the algorithm computes a bipartitioning with the lowest communication volume out of all possible bipartitionings that obey the load balance constraint. The algorithm is based on partitioning the rows and columns of the matrix into three sets instead of partitioning its nonzeros into two sets. To investigate the performance of our branch-and-bound algorithm, we applied it to all matrices of the University of Florida sparse matrix collection with up to 1000 nonzeros. For 85% of the matrices, the optimal communication volume was found within a single day of computation, and for 58% even within a second.

The gap we found between heuristic and optimal solutions suggests that improvement of the heuristics is possible in the future. By investigating certain matrices and their optimal solutions in detail, it may be possible to find directions in which to improve the heuristics, which is subject to further research. The comparison with the 1D localbest method shows that 1D bipartitionings generally have a much higher communication volume than the optimal volume, indicating that optimal bipartitionings are often 2D.

For future work, it would be interesting to compare the actual timings of a parallel sparse matrix–vector multiplication for optimal and heuristic partitionings. This may require further improvement of our exact algorithm, extending it to handle larger problems, hopefully reaching realistic application problem sizes. This would demonstrate the impact of communication volume reduction by better partitioning, and would also provide insight into the limitations of communication volume as a metric.

The computation time of the branch-and-bound method depends highly on the quality of the lower bounds on the communication volume of partial solutions. Therefore, by using a better lower bound, the computation time of our method might be improved. This would also result in the ability to optimally bipartition larger matrices. Whether better partitioning strategies or better practical lower bounds exist is subject to further research. One possibility of solving larger problems would be to parallelize the branch-and-bound method, assigning subtrees to the processors of the parallel computer used, as we did for an earlier version of our implementation [33].

A generalization of our branch-and-bound method is the case of more than two processors. In this case, it is still possible to partition the rows and columns of the matrix instead of its nonzeros. When partitioning for p

processors, this approach would generate a search tree with $2^p - 1$ children at every node, and a total number of possible partitionings of $(2^p - 1)^{m+n}$. Although this number may seem prohibitively large for $p > 2$, it might be possible to prune the search tree in a better way, for instance because larger variations in communication volume caused by a single row or column may lead to stronger lower bounds.

References

[1] D. A. Bader, H. Meyerhenke, P. Sanders, D. Wagner (Eds.), Graph Partitioning and Graph Clustering, Vol. 588 of Contemporary Mathematics, AMS, Providence, RI, 2013.

[2] C. Berge, Two theorems in graph theory, Proceedings National Academy of Sciences USA 43 (9) (1957) 842–844.

[3] R. H. Bisseling, J. Byrka, S. Cerav-Erbas, N. Gvozdenović, M. Lorenz, R. Pendavingh, C. Reeves, M. Röger, A. Verhoeven, Partitioning a call graph, in: Proceedings Study Group Mathematics with Industry 2005, Amsterdam, CWI, Amsterdam, 2005, pp. 95–107.

[4] R. H. Bisseling, B. O. Fagginger Auer, A. N. Yzelman, T. van Leeuwen, Ü. V. Çatalyurek, Two-dimensional approaches to sparse matrix partitioning, in: U. Naumann, O. Schenk (Eds.), Combinatorial Scientific Computing, Chapman & Hall / CRC Press, 2012, pp. 321–349.

[5] E. G. Boman, Ü. V. Çatalyürek, C. Chevalier, K. D. Devine, The Zoltan and Isorropia parallel toolkits for combinatorial scientific computing: Partitioning, ordering and coloring., Scientific Programming 20 (2012) 129–150.

[6] E. G. Boman, K. D. Devine, S. Rajamanickam, Scalable matrix computations on large scale-free graphs using 2D graph partitioning, in: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, SC '13, ACM, New York, NY, USA, 2013, pp. 50:1–50:12.

[7] J. A. Bondy, U. S. R. Murty, Graph Theory, Vol. 244 of Graduate Texts in Mathematics, Springer, 2008.

[8] A. Buluç, H. Meyerhenke, I. Safro, P. Sanders, C. Schulz, Recent advances in graph partitioning, in: Algorithm Engineering, survey collection for the DFG SPP 1307, LNCS, vol. 9220, Springer, 2015, in press.

[9] A. E. Caldwell, A. B. Kahng, I. L. Markov, Optimal partitioners and end-case placers for standard-cell layout, IEEE Trans. on CAD of Integrated Circuits and Systems 19 (11) (2000) 1304–1313.

[10] Ü. V. Çatalyürek, C. Aykanat, Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication, IEEE Transactions on Parallel and Distributed Systems 10 (7) (1999) 673–693.

[11] Ü. V. Çatalyürek, C. Aykanat, A fine-grain hypergraph model for 2D decomposition of sparse matrices, in: Proceedings of the 15th International Parallel & Distributed Processing Symposium, IEEE Computer Society, Washington, DC, USA, 2001, p. 118.

[12] Ü. V. Çatalyürek, C. Aykanat, B. Uçar, On two-dimensional sparse matrix partitioning: Models, methods, and a recipe, SIAM Journal on Scientific Computing 32 (2) (2010) 656–683.

[13] Ü. V. Çatalyürek, M. Deveci, K. Kaya, B. Uçar, UMPa: A multi-objective, multi-level partitioner for communication minimization., in: Graph Partitioning and Graph Clustering, AMS, 2012, pp. 53–66.

[14] C. Chevalier, F. Pellegrini, PT-Scotch: A tool for efficient parallel graph ordering, Parallel Computing 34 (6-8) (2008) 318–331.

[15] T. A. Davis, Y. Hu, The University of Florida sparse matrix collection, ACM Trans. Math. Softw. 38 (1) (2011) 1:1–1:25.

[16] D. Delling, D. Fleischman, A. V. Goldberg, I. Razenshteyn, R. F. Werneck, An exact combinatorial algorithm for minimum graph bisection, Mathematical Programming (2014) 1–42.

[17] K. D. Devine, E. G. Boman, R. Heaphy, R. H. Bisseling, U. V. Catalyurek, Parallel hypergraph partitioning for scientific computing, in: Proceedings IEEE International Parallel and Distributed Processing Symposium 2006, IEEE Press, Los Alamitos, CA, 2006, p. 102.

[18] E. D. Dolan, J. J. Moré, Benchmarking optimization software with performance profiles, Mathematical programming 91 (2) (2002) 201–213.

[19] A. Felner, Finding optimal solutions to the graph partitioning problem with heuristic search, Annals of Mathematics and Artificial Intelligence 45 (3–4) (2005) 293–322.

[20] C. M. Fiduccia, R. M. Mattheyses, A linear-time heuristic for improving network partitions, in: Proc. 19th IEEE Design Automation Conference, IEEE Press, Los Alamitos, CA, 1982, pp. 175–181.

[21] W. W. Hager, D. T. Phan, H. Zhang, An exact algorithm for graph partitioning, Mathematical Programming 137 (1-2) (2013) 531–556.

[22] B. A. Hendrickson, R. Leland, An improved spectral graph partitioning algorithm for mapping parallel computations, SIAM Journal on Scientific Computing 16 (2) (1995) 452–469.

[23] S. E. Karisch, F. Rendl, J. Clausen, Solving graph bisection problems with semidefinite programming, INFORMS Journal on Computing 12 (3) (2000) 177–191.

[24] G. Karypis, V. Kumar, A fast and high quality multilevel scheme for partitioning irregular graphs, SIAM Journal on Scientific Computing 20 (1) (1998) 359–392.

[25] G. Karypis, V. Kumar, Multilevel k-way hypergraph partitioning, in: Proceedings 36th ACM/IEEE Conference on Design Automation, ACM Press, New York, 1999, pp. 343–348.

[26] G. Karypis, V. Kumar, Parallel multilevel k-way partitioning scheme for irregular graphs, SIAM Review 41 (2) (1999) 278–300.

[27] E. Kayaslaan, A. Pinar, Ü. V. Çatalyürek, C. Aykanat, Partitioning hypergraphs in scientific computing applications through vertex separators on graphs, SIAM Journal on Scientific Computing 34 (2) (2012) A970–A992.

[28] B. W. Kernighan, S. Lin, An efficient heuristic procedure for partitioning graphs, Bell System Technical Journal 49 (1970) 291–307.

[29] D. Kucar, S. Areibi, A. Vannelli, Hypergraph partitioning techniques, Dynamics of Continuous, Discrete and Impulsive Systems Series A: Mathematical Analysis 11 (2–3a) (2004) 339–367.

[30] A. H. Land, A. G. Doig, An automatic method of solving discrete programming problems, Econometrica 28 (3) (1960) pp. 497–520.

[31] T. Lengauer, Combinatorial algorithms for integrated circuit layout, John Wiley and Sons, Chichester, UK, 1990.

[32] F. Pellegrini, J. Roman, Scotch: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs, in: HPCN Europe, 1996, pp. 493–498.

[33] D. Pelt, Matrix partitioning: Optimal bipartitioning and heuristic solutions, Master's thesis, Utrecht University (2010).

[34] D. M. Pelt, R. H. Bisseling, A medium-grain method for fast 2D bipartitioning of sparse matrices, in: Proceedings IEEE International Parallel and Distributed Processing Symposium 2014, IEEE Press, 2014, pp. 529–539.

[35] P. Sanders, C. Schulz, Think Locally, Act Globally: Highly Balanced Graph Partitioning, in: Proceedings of the 12th International Symposium on Experimental Algorithms (SEA'13), in: LNCS, vol. 7933, Springer, 2013, pp. 164–175.

[36] N. Sensen, Lower bounds and exact algorithms for the graph partitioning problem using multicommodity flows, in: Algorithms - Proceedings ESA 2001, in: LNCS, vol. 2161, Springer, 2001, pp. 391–403.

[37] A. Trifunović, W. J. Knottenbelt, Parallel multilevel algorithms for hypergraph partitioning, Journal of Parallel and Distributed Computing 68 (5) (2008) 563–581.

[38] B. Vastenhouw, R. H. Bisseling, A two-dimensional data distribution method for parallel sparse matrix-vector multiplication, SIAM Review 47 (1) (2005) 67–95.

[39] C. Walshaw, M. Cross, JOSTLE: Parallel Multilevel Graph-Partitioning Software – An Overview, in: F. Magoules (Ed.), Mesh Partitioning Techniques and Domain Decomposition Techniques, Civil-Comp Ltd., 2007, pp. 27–58.